SOFTWARE DESIGN DOCUMENT

FOR THE

AIRBORNE BROADCAST INTELLIGENCE (ABI) SYSTEM

Contract Number F19628-95-C-0143

6 March 1998

Prepared for:
Electronic Systems Center
Air Force Materiel Command
50 Griffiss Street
Hanscom Air Force Base, Massachusetts 01731-1619

Prepared by:

Lockheed Martin Command Control Systems
9970 Federal Drive
Colorado Springs, CO 80921

This Page Intentionally Left Blank.

TABLE OF CONTENTS

LIST OF APPENDICES

LIST OF TABLES

LIST OF FIGURES

This Page Intentionally Left Blank.

1.0 Scope

1.1 Identification

This Software Development Design Document (SDD) specifies the Computer Software Configuration Item (CSCI) design for the Airborne Broadcast Intelligence (ABI) prototype system. This SDD describes in detail each of the major functional areas, the architecture of the software, the data structures, and all associated processes.

1.2 System Overview

The ABI system consists of numerous software programs, running in concert to provide a situation awareness system that requires little operator setup and no operator interaction during use. The system follows the user around the world, using the attached Global Positioning System (GPS) receiver to locate where in the world the user is, and displays the maps, charts, and country outlines that the operator has selected in the scales and orientation selected. (GPS functionality is for AMC only). External events and threat definition messages are received by the system, correlated, and displayed on maps selected by the operator so that the situation can be visually assessed.

The ABI system consists of components from the Combat Intelligence System (CIS) Automatic Associator (AA) 5.0 baseline and the Multi Source Tactical System (MSTS). The CIS AA 5.0 components will be renamed and referred to as the ABI Data Fusion Engine, and the MSTS components will be renamed and referred throughout this document as the ABI Displays. Sections 4.1 through 4.6 explain the ABI Data Fusion Engine and Sections 4.7 through 4.28 explain the ABI Displays .

1.3 Purpose

This SDD is written with the objective of explaining what each CSCI does, how the components interrelate, and describe the software design and high levels of the ABI system. This document will provide other professionals an understanding of the system and assist in future development efforts.

1.4 Document Overview

This document is organized as follows:

Section 1, Scope, consists of the system identification, a brief description of the ABI system, and a summary of the purpose and contents of this SDD.

Section 2, Referenced Documents, contains a list of documents referenced in this SDD.

Section 3, Design Overview, describes the overall architecture of the ABI system.

Section 4, Detailed Design, describes the functional area, architectural design, and processing of each functional area.

Section 5, Acronyms.

Appendix A, Requirements Traceability, contains a traceability matrix which serves as the requirements baseline for the development of the ABI system.

2.0 Referenced Documents

2.1 Government Documents

This section lists by document number, title, date, and classification all documents referenced in preparing this SDD. It also identifies the source of all documents not available through normal Government stocking activities.

a. ABI-U-NA-N-001, Software Test Description for the Airborne Broadcast Intelligence (ABI) System, 19 January 1998 (Revision 4); Electronic Systems Center, Air Force Materiel Command, 50 Griffiss Street, Hanscom Air Force Base, Massachusetts 01731-1619. (U)

b. ABI-U-NA-N-002, Software User's Manual for the Airborne Broadcast Intelligence (ABI) System (Draft), 15 August 1997; Electronic Systems Center, Air Force Materiel Command, 50 Griffiss Street, Hanscom Air Force Base, Massachusetts 01731-1619. (U)

c. DOD-STD-2167A, Defense System Software Development, 29 February 1988; Department of Defense, Washington, DC 20301. (U)

d. DIAM 50-4, Security of Compartmented Computer Operations, 24 June 1980, Defense Intelligence Agency, Washington, DC 20340-4032. (C)

e. JCS PUB 6-04.22 (I-O) and JCS PUB 6-04.23 (P-Z), U. S. Message Text Formatting, (USMTF) Program, USMTF Message Preparation Instructions, 1 October 1990; Defense Intelligence Agency, Washington, DC 20340-4032. (U)

f. Statement of Work for Airborne Broadcast Intelligence (ABI) Theater Battle Management Core Systems (TBMCS), 17 March 1997; Lockheed Martin Command and Control Systems, Colorado Springs Division, 9975 Federal Drive, Colorado Springs, Colorado, 80921. (U)

2.2 Non-Government Documents

a. Motif Style Guide, Open Software Foundation, Incorporated, 1991; Prentice-Hall, Incorporated, Englewood Cliffs, New Jersey 07632.

b. OSF/Motif Documentation - Figures, 1990; OSF Corporation, Cambridge, Massachusetts 02142. (U)

3.0 Design Overview

The ABI System Architecture as shown in Figure 3.0-1, integrates strategic intelligence, tactical intelligence, and GPS broadcasts to enhance real time threat awareness/avoidance. (GPS is used for AMC only). ABI serves a wide user base and supports flexible mission demands. ABI uses the CIS AA as its correlator and integrates the look and feel of MSTS.

The ABI system is a situation awareness capability designed to receive, process and display real-time intelligence and operational information overlaid onto imagery and charts. The technology includes flight following, two and three-dimensional threat displays, terrain perspective views and mission preview.

The system loads and stores aeronautical charts, multispectral and high-resolution imagery. Near real-time Signal Intelligence (SIGINT) and Radio Detection and Ranging Intelligence (RADINT) is received in-flight and its symbology is overlaid onto stored images and charts, indicating parameters and lethality ranges in two and three dimensional representations. Flight following includes GPS input/update. Off-line mission rehearsal fly-through can be generated as can interactive, operator-controlled fly-over.

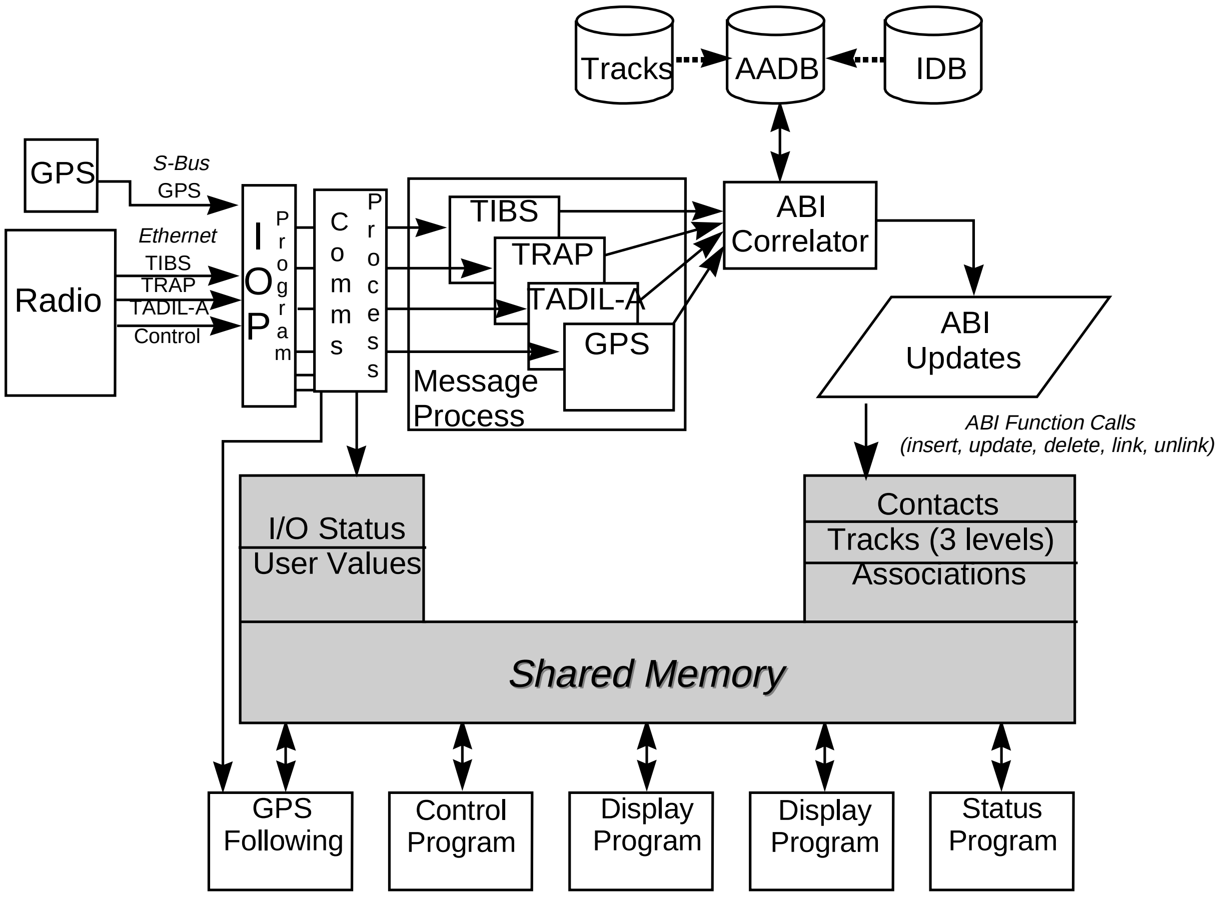


Figure 3.0-1 ABI System Architecture

3.1 Hardware Suite

The ABI hardware suite consists of a single processor Sun Ultra 2 workstation with the Creator 3D Graphics System.

Additional hardware consists of:

QuadZebra radio or MATT radio

Supporting external cryptos to include two KGR-96 and a KG-40A (to be enabled internally in fiscal year 98)

Crypto key mats for appropriate inputs

Crypto fill device CZY-10, KIK-13, or KOI-18

3.2 ABI Software

The ABI software consists of the following:

CIS 1.2 / AA 5.0

ABI Database (AA Build 22.4 DB)/SYBASE IDB

Solaris 2.5.1 (May 1996)

ABI Display Software (MSTS)

Modular UNIX code, C-language, OSF/Motif

4.0 Detailed Design

The ABI system consists of a correlation engine, visualization software, configuration and control software, a collection of data files (world data bank outlines of countries ADRG chart files, imagery files, Area of Interest (AOI) definition files, route definition files, and satellite ephemeris data files). Each ABI system running on a workstation also has a collection of private data files describing how the operator has setup the ABI visualization (which symbols, routes, AOIs are visible) and user preferences (line widths, symbol sets, units of measure). The visualization data is also kept in the memory of each workstation, in a shared memory segment that all programs running on that workstation can read and update. The shared memory segment allows for well integrated visualization operations that are very fast.

This detailed design describes each major functional program that is a part of ABI. The order of the ABI Displays programs, discussed in Sections 4.7 through 4.28, are in the order that the programs are started on the ABI system or in the control banner pull down.

4.1 ABI Runtime Environment Functional Area Design

Sections 4.1.1 through 4.1.2.5 describe the required modifications to the operational environment to incorporate the CIS/AA 1.2 baseline with MSTS.

4.1.1 ABI Runtime Environment Architecture

4.1.1.1 Setup Receiver Control Output Monitoring

The MSTS Communications process will send receiver output to AA via a socket interface. The Local Area Network (LAN) interface software is already available in the AA Comms CSCI to leverage for this functionality in ABI.

4.1.1.2 Solaris 2.5.1

The CIS 1.2 platform is a Solaris 2.5.1 compilation of AA code currently performed on a Sparc 20 or better machine using Triteal CDE.

4.1.2 MSTS Under the CIS Umbrella

The processes described in Sections 4.1.2.1 through 4.1.2.4 are handled by CIS/AA.

4.1.2.1 Initialization/Startup

4.1.2.1.1 Initialization

- The normal AA background processes will be started at boot.
- The MSTS /abi/runtime/bin/iop process which configures shared memory and the Quad Z or generic MATT radio receiver and GPS should be started at boot or via the sys_exec.
- The auto_purge processes will be disabled from the sys_exec. Track removal will be done through MSTS.

4.1.2.1.2. Startup

The MSTS display process and the AA process that controls the interface to MSTS will be started upon operator selection of the ABI icon.

4.1.2.2 Login/Security

Login is disabled.

4.1.2.3 Auto-Upload of Snapshot DataBase

ABI Database Transfer Procedures have been developed to transfer data from a CIS/AA ground station and copy the files onto an ABI system. This procedure will also work from one ABI System to another ABI. This includes transfer of an Integrated Database (IDB) and/or Near-Real Time (NRT) database.

4.1.2.4 GPS System Time synch (AMC Only)

A modification of the system time should not cause any interruption of AA processing. However, Time of Receipt (TOR) for contacts created prior to the GPS sync may not reflect the true time of creation. If any time discrepancies occur during track creation, the AA autopurge daemon may prematurely delete tracks which are not as old as they appear to be. This problem may be remedied by either synchronizing time prior to running in messages or the operator may disable the autopurge capability.

The design approach for system time synchronization will be that the AA Autopurge function will be disabled and that the system clock will be set via MSTS to the initialization time of the system.

4.1.2.5 Error Handling

Error handling will not be changed from CIS/AA.

4.2 Performance Track Processor Cache Functional Area Design

The Track Processor will be modified to retain pertinent facility data in memory. When facility data is needed by the Track Processor, memory will be accessed rather than accessing the Facility database.

4.2.1 Performance Track Processor Cache Architecture

The Performance Track Processor Cache (PERF-TP CACHE) process will read all required facility data and store it in ordered lists in memory at startup.

4.2.2 PERF-TP CACHE Processing Description

The following sections describe the PERF-TP CACHE processing. This description is broken down into a section on data flow, initialization, inter-process communications, and error handling.

4.2.2.1 Data Flow

The XIDB tables and fields required to perform facility association will be read by the Track Processor. If caching is turned on, the Track Processor will read all the required fields from the facility tables (XIDBF, XIDBFQL, and MIDBFQL_ELNOT) from either the real or exercise IDB. The Track Processor will also read all Template and Template Equip records from the Near Real Time Support (nrt_support) Database. These records will be placed in ordered lists. The lists will be ordered on either the primary key for the table which uniquely identifies the record or a secondary key used for fast access. When the Track Processor requires a particular record or set of records, either a database query will be performed using the object library, or if caching is turned on, the appropriate cache get function will be invoked.

4.2.2.2 Initialization

The Track Processor, based on a command line argument of “-cache on”, will read all required facility data and store it in ordered lists in memory at startup.

4.2.2.3 Inter-process Communications

No new IPC mechanisms are required to support this functional area.

4.2.2.4 Error Handling

All encountered errors are printed to STDERR which is directed to a log file. Depending on the gravity of the error, the Track Processor may terminate after logging the error.

4.3 User Interface Functional Area Design

The starting point for the ABI User Interface (UIF) functional area is the Window Manager currently used for TBMCS/CCS (CIS Core Software). The CDE/Triteal is the Window Manager for CCS and is the basis for further user interface development for ABI.

4.3.1 User Interface Architecture

The current User Interface for CIS under CDE/Triteal is the Application Manager window. This window consists of folders that the operator can open to gain access to system foreground software components. In order to provide access to ABI, an icon (Start ABI) will be added to the CCS window management structure. The Start ABI icon provides the one-button mechanism for bringing up the MSTS windows.

3D displays/maps will be integrated into the ABI system (MSTS).

ABI Database Transfer Procedures have been developed to transfer data from a CIS/AA ground station and copy the files onto an ABI system. This procedure will also work from an ABI System to another ABI. This includes an Integrated Database (IDB) and Near-Real Time (NRT) data base.

4.3.2 UIF Functional Area Processing Description

The following paragraphs describe the UIF processing. This description is divided into a description of data flow, initialization, inter-process communications, and error handling.

4.3.2.1 Data Flow

In order to understand the User Interface flow, it must be understood that the ABI system will function most efficiently with initial IDB and NRT databases loaded. These databases will be obtained from any ground CIS/AA system or a ground ABI system. Once an ABI system has had the databases loaded, it will be up to the user to reload new IDB and NRT data on future restarts.

NOTE
Loading of the databases is optional, the system can function without these but only the current information being received by the system will be available.

When the IDB and NRT loads have been completed, ABI will begin by starting the process that controls the AA to MSTS data flow (e.g., ABI Display Communicator) and the MSTS process. Then, the classic MSTS displays will appear, covering the CDE "control panel." When MSTS is exited, the CDE panel will again be visible, along with the open ABI folder that was there when START_ABI was executed.

The following is an outline of the steps the user must follow in order to start the ABI system.

<u>CIS/AA only</u> - Select "On" position on power switch.

<u>Loading IDB and NRT data</u> - Extract IDB and NRT databases from a ground CIS/AA workstation then load onto ABI System. NRT data should be loaded first because the system will require a reboot. If the IDB is loaded and the system subsequently rebooted, the IDB data will reflect old, outdated information. A 3.5" floppy disk in UNIX format containing the ABI load/extract scripts, and a writeable DAT tape are required. These are classified items will be maintained IAW approved classified material handling procedures established by Government users.

<u>ABI</u> - The ABI program is initiated by selecting the *ABI* icon.

4.3.2.2 Initialization

The `sys_exec` modules `sysMain.c` and `sysSignal.c` will be modified to include new signals to the `sys_exec` process for stopping all background processes and restarting them. The new signals are `SIGTSTP` and `SIGCONT` which map to the functions `sig_stop_fun` and `sig_continue_fun`, respectively.

A new mechanism for signaling the `start_abi_exec` background processes will be introduced to determine if a process is currently running. The "signal" is an OM publish event. The `start_abi_exec` subscribes to events of a new "dummy" table in the `aa_support` database (table definition: `StartAbi.def`). When a user double clicks the Start_ABI icon, the `start_abi_client` process issues a publish event for the Start_ABI dummy table. The background process `start_abi_exec` receives notification of this and starts the `start_abi` process. The `start_abi` process then makes root-user privileged calls to determine if the `abi_data_exchange` and MSTS process are already running and only starts them if they are not. This ensures only one `abi_data_exchange` and one MSTS process will be running at any one time. Since the commands are privileged, the `start_abi_exec` background process will be needed and started by the `sys_exec` process and hence inherit the root-user id. In turn, the `start_abi` process will then also inherit root-user privileges when started by the `start_abi_exec`.

4.3.2.3 Inter-process Communications

Existing CDE/Triteal Window Manager Inter-process communications will be retained. Two new signals will be added as described in section 4.3.2.2. These signals will be used by Start_ABI to request a stop and restart to sys_exec of the background processes.

4.3.2.4 Error Handling

Error handling will include the normal methods of `fprintf` to `stderr` as well as a pop-up information window to the user when necessary. If an error occurs the user is notified, and given the opportunity to continue ABI. For example, if the `tar` program does not exist so that the user cannot "un-tar" the NRT files on tape, the user may wish to continue by using the NRT data that exists already on disk.

4.3.3 UIF Functional Area Changes

The following sections describe the items that are changed in order to meet the requirements of this functional area.

4.3.3.1 New Software Libraries

New libraries will consist of new icon bitmap files, as well as a new binary repository for ABI-specific executables. The new module that will control the starting of ABI will be `start_abi.c`. The new script `idb_load` will read IDB extract files from tape and copy them to the `/usr/IDBextract` directory on disk so that these files may be read by the DD Controller which will be subsequently forked by the `idb_load` script upon completion of the tape read.

The new script `nrtload` will be used on a CIS host to obtain a tape archive of NRT data which may then be auto-uploaded on an ABI host upon selection of the Start ABI icon.

4.3.3.2 Modified Software Libraries

The libraries that will be modified are the Window Manager configuration files.

4.3.3.3 New User Interface Windows

ABI/CIS will already be running on top of the Window Manager (CDE/Triteal), which runs on top of Motif/X, which runs on the hardware (Creator/3D). The CIS Man Machine Interface (MMI) will provide another Application_Manager icon called Start_ABI. Originally, the Application_Manager was to be used for the location of the two new icons, Start ABI and IDB Load. However, the Application Manager could not be modified, therefore the Start_ABI button will be moved to the AA

Window of the INTEL_Applications folder accessed via the Application Manager window. The icons can then be dragged onto the desktop where they will remain.

4.3.3.4 Modified User Interface Windows

The AA Window includes an ABI startup icon.

The MSTS Executive Window may be modified to provide a drill-down button to return to the CIS/AA screen, although this is also accomplished by exiting the MSTS. To provide all of the icons the files listed in Table 4.3.3.4-1, New and Modified CIS Desktop Files, were changed or added:

Table 4.3.3.4-1 New and Modified CIS Desktop Files

New	Changed
/usr/l/conf/aa_client/dt/appconfig/icons/C/StartAbi.m.pm /usr/l/conf/aa_client/dt/appconfig/icons/C/StartAbi.t.pm medium and small icons for Start Abi	/usr/l/share/tip_menus/menus/menu.config Add the two new icons to the AA window menu.
/usr/l/conf/aa_client/dt/appconfig/icons/C/Idb_load.m.pm /usr/l/conf/aa_client/dt/appconfig/icons/C/Idb_load.t.pm medium and small icons for IDB Load	/usr/l/conf/aa_client/binaries.doc List idb_load and start_abi as client icons
aa_client/solaris/bin/idb_load Invoked by the icon to tar the idb from tape and invoke the DD Controller process	/usr/l/conf/aa_client/dt/appconfig/types/C/aa_client.dt Add the parameters of the two new icons and executables

4.4 Data Base-AA/MSTS Interface Functional Area Design

4.4.1 Data Base-AA/MSTS Architecture

Figure 4.4.1-1, Database (DTB)-AA/MSTS Interface, is a high-level data flow diagram depicting the DTB-AA/MSTS Interface. ABI will receive data transmissions from a receiver, parse the messages in message processing and send the parsed messages to the AA Correlator. The AA Correlator will then correlate the messages and send the results to the database and to shared memory. The Track Processor and TTA processes will provide further correlation and write to the database and shared memory. AA display processes and autopurge (if not disabled) will update the database, and the database will send update notifications to the correlation processes. Data written by AA to shared memory will be read by MSTS and then used to update the MSTS displays. MSTS will also send delete requests to AA via API calls. MSTS will call an API that will utilize an ipc call to perform the delete function. A new process that will handle all the add and update notifications from the database will invoke the API calls. This method has the disadvantage of additional ipc calls needed for database notfication, but has the advantage of only one process writing to shared memory. As a further means of streamlining, the interface will only send data to MSTS that has changed as opposed to sending the entire data record. This will reduce the amount of data sent to shared memory since many updates only entail a change to one or two fields out of the dozens of fields in a record.

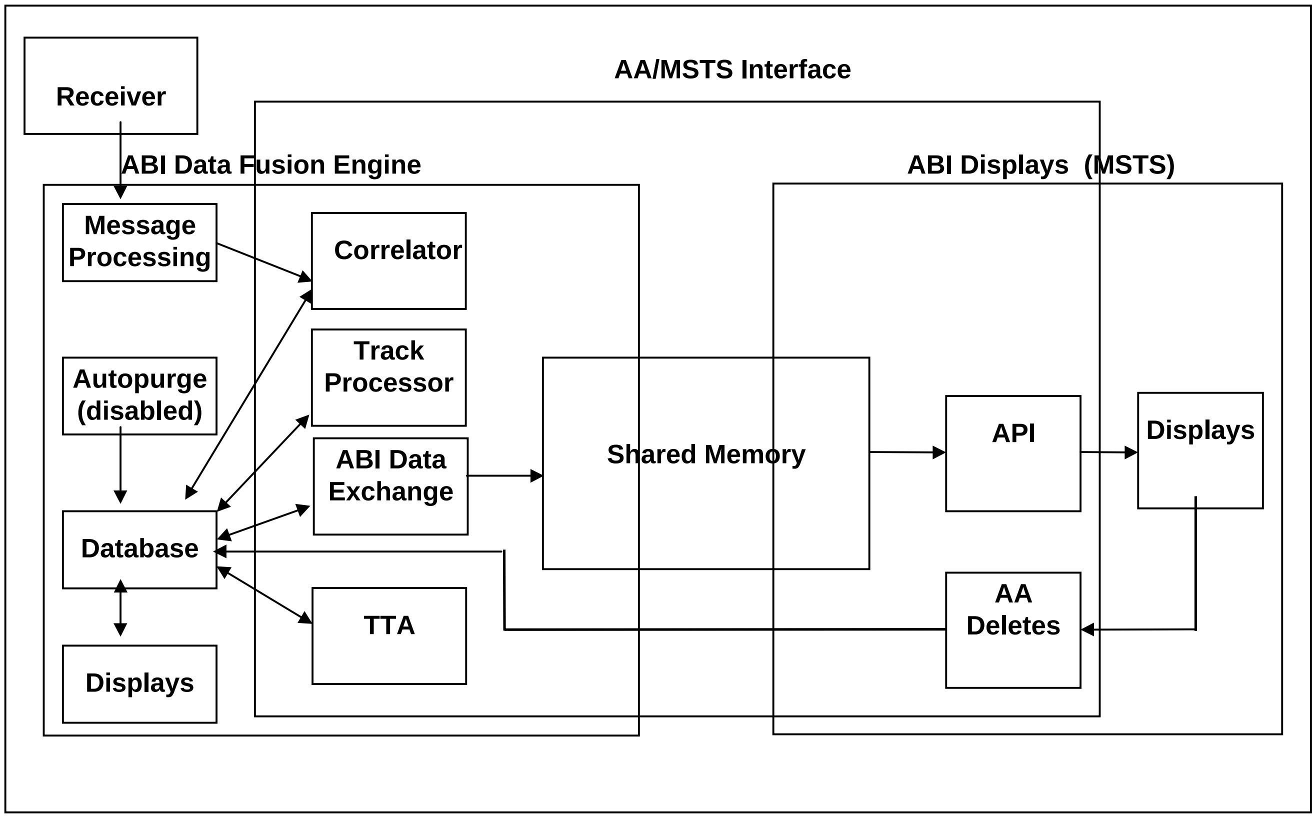


Figure 4.4.1-1 DTB-AA/MSTS Interface

4.4.2 DTB-AA/MSTS Interface Processing Description

The following paragraphs describe the DTB-AA/MTS Interface processing. This description is divided into a description of data flow, initialization, inter-process communications, and error handling.

4.4.2.1 Data Flow

Contact data is parsed from a receiver and sent to the correlator process. The correlator process creates and updates contacts, groups and tracks and sends the data to the database. The TIBS/TADIL Associator (TTA) provides correlation for air tracks and the Track Processor provides further correlation; both of which also send data to the database. User-edited contacts are sent to the correlator for recorrelation and are then sent to the database. Delete requests are specified by the user or generated by autopurge (if and when enabled - however it is recommended autopurge be disabled for ABI and sent to the database. All data sent to MSTS will be through API calls. Data sent to MSTS will be retrieved, processed, and displayed by MSTS processes. MSTS sends delete requests to AA using an API call. AA will delete the data from the database and delete notifications will be sent to the registered processes.

Each new or updated correlated contact, group or track will be converted from an Object Manager (OM) record type into a data structure that will be written to shared memory. This data structure will include the field identifier and value for each new or updated field. The ABI Data Exchange will only write to shared memory the fields that are new to the record or have been updated. In order to determine which fields are updated, a call will invoke an OM function that returns a status array indicating which fields have changed. The ABI Data Exchange will determine which fields have changed; and for each field add the field identifier and field value to a list of structures and maintain a count of the total number of changed fields.

Once all fields and values are added to the list, the list and count total will be used as arguments in the API function call. Additional arguments will be the type of record (e.g., Contact, Track, etc.) and the record id. These functions will write the record to shared memory where MSTS will then retrieve and process the record. In the case of a delete, only the record type and record id will be specified in the API call. In addition, a link and unlink call will be used in the API to specify when a contact is added (linked) or unlinked to a group, and when a group is linked or unlinked to a track. The API calls will be invoked from a separate process that will handle all of the API calls to MSTS.

4.4.2.2 Initialization

The ABI Data Exchange will be initialized when invoked by the Start_ABI icon and background process. This process will ensure that the ABI Data Exchange is not already running. If not, it will start the ABI Data Exchange process which will first read all of the data (contact, tracks, groups, etc.) in the database and pass the data up to MSTS. It will then register with the Object Manager Broker for all tables of the NRT database that it wishes to be notified of when a change occurs. Currently these are the Trackset, Contact, Contact Group and link tables.

4.4.2.3 Inter-Process Communications

All AA/MSTS Interface communication is through shared memory and files. If there are other client machines, MSTS will handle the dissemination of AA data to the other clients through ipc socket calls. Shared memory was selected based on the fact that Boeing already uses the shared memory methodology to transfer data, and also because shared memory is relatively a fast method for data transfer compared to sockets.

4.4.3 Error Handling

Currently the API functions defined by MSTS return type void so there is no indication if the function succeeded. Error handling will be added to check that conversion from an OM record to the API data structure was done successfully.

4.4.4 DTB-AA/MSTS Interface Functional Area Changes

The following sections describe the items that are changed in order to meet the requirements of this functional area.

4.4.4.1 New Software Libraries

The two libraries listed below will be created to support this functional area.

- MSTS to AA library (libabiexchange.a) - This library will process delete requests received from MSTS. The library may eventually be expanded in the future to include additional requests such as data additions or updates.
- MSTS API library (libabi.so.1) - This library contains functions utilized by AA to send data to MSTS; functions included are ones to add, update, delete and link data.

4.4.4.2 Modified Software Libraries

The libraries that will be modified are the Window Manager configuration files.

4.4.4.3 New User Interface Windows

No new user interface windows will be developed under this version of ABI.

4.4.4.4 Modified User Interface Windows

The Correlator disk library (libdsk.a) will be modified to include the addition of routines to convert OM records to MSTS API structures needed in the API calls.

4.5 Database - Integrated Data Base Snapshot Functional Area Design

The Database (DB) Integrated Database (IDB) Snapshot functionality will provide a streamlined approach in the transfer of IDB (facility) data from a CIS IDB maintained as a SYBASE database to an ABI IDB SYBASE database. This functionality will also optimize the access of the IDB data by

AA to help meet performance requirements. This performance related enhancement is accomplished by code modifications to the track processor.

4.5.1 IDB Snapshot Architecture

The IDB Snapshot functional area software consists of the following components:

- IDB Load script
- System Startup Configuration File

4.5.2 DB-IDB Snapshot Processing Description

The following paragraphs describe the DB-IDB Snapshot processing. This description is divided into data flow, initialization, inter-process communications, and error handling.

4.5.2.1 Data Flow

The input data source for this functional area is the CIS SYBASE IDB and the output data destination is the ABI SYBASE IDB.

The DB-IDB Snapshot functional area software consists of the new script, idb_load, a modification to the system startup file, sysConfig.dat, and code modifications to the Track Processor application. The following paragraphs provide implementation details.

- IDB Load script - A new c-shell script, idb_load, reads a UNIX tar tape. This tape contains IDB extract information and is created on a CIS machine which contains a populated IDB for the area of interest. The new script will place the IDB extract files in a temporary hold directory, /usr/IDBextract.

- System Startup Configuration File - The system startup configuration file ($BTG_SYSTEM/solaris/config/sysConfig.dat) used by sys_exec will be modified to include the “-cache on” option during the startup of the track processor.

- Track Processor - The Track Processor associates NRT (tracks and contact groups) data to IDB (facilities and facility equipment) data. It will be modified to run with an optional caching (“-cache on”) argument which will cause all pertinent IDB data (i.e., the requisite tables and subset of their fields) to be read along with the complete set of Template data from the NRT Support database. Subsequent queries of this data will merely require memory accesses instead of database accesses thereby increasing processing speed of this application and optimizing the access of IDB data.

4.5.2.2 Initialization

An initialization process is not required for this functional area.

4.5.2.3 Inter-Process Communications

The existing ipc mechanisms were retained.

4.5.3 Error Handling

The existing error handling methodology was retained.

4.5.4 Functional Area Changes

The following sections describe the changes that will be made to the system libraries and/or interface windows.

4.5.4.1 New Software Libraries

No new software libraries were created.

4.5.4.2 Modified Software Libraries

No new software libraries were created.

4.5.4.3 New User Interface Windows

The IDB Controller window was created for this functional area.

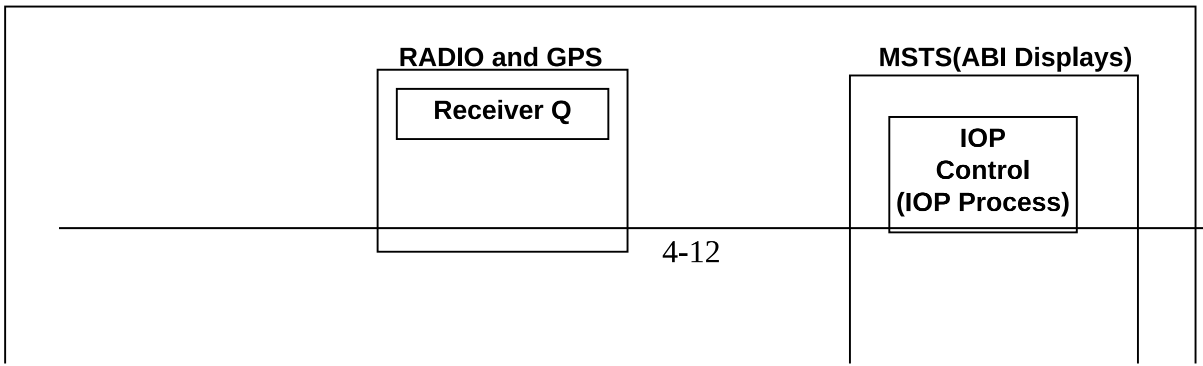
4.5.4.4 Modified User Interface Windows

No user interface windows were modified.

4.6 Message Design

4.6.1 Message Functional Area Architecture

ABI Architecture



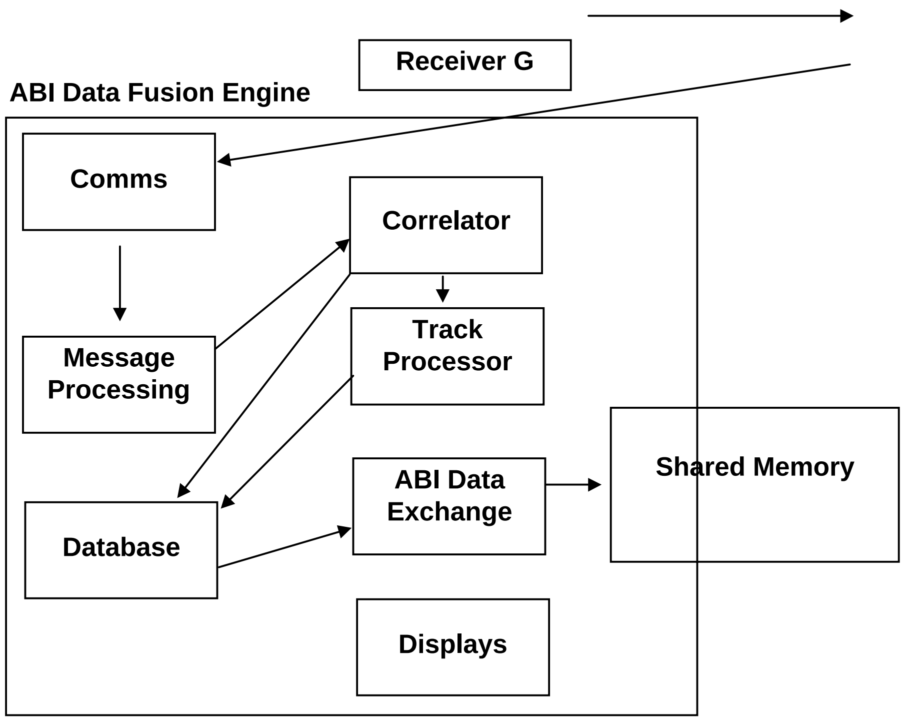


Figure 4.6.1-1 ABI Architecture

Figure 4.6.1-1, ABI Architecture, shows the part of the ABI system that the Message functional area operates under.

The AA Comms Config software must be configured and the lines enabled for the input that will be received from the INTEL Radio receiver via the Input/Output Processor (IOP) operation. As shown in Figure 4.6.1-1, the IOP function is controlled by the MSTS (ABI Displays) software and it controls the input lines directly from the radio and the GPS receiver.

The message process will receive data from the Intel Radio, including TDDS, TOPS, and TIBS messages. The message process will parse each message into a new contact and send the contact to the correlator which correlates the contact and sends the correlated data to shared memory. This is actually accomplished via Object Manager Broker notification to a new process (ABI Data Exchange) which then calls API functions that access shared memory.

4.6.1.1 Data Flow

The QuadZ (AMC Only) input lines are controlled by the IOP and Comms Config processes. The IOP controls what is received from the radio, and the Comms Config controls what the ABI Data Fusion Engine receives. The ABI Data Fusion Engine will receive data from the receiver on a LAN line, including TDDS, TOPS, TIBS and TADIL-A messages. The message process will parse each message into a new contact and send the contact to the correlator which correlates the contact and updates the database. The ABI Data Exchange process is then notified of the update and sends the correlated data to shared memory.

The functionality to process TDDS, TOPS, TIBS and TADIL data messages over a LAN already exists in the AA software but must be verified with the new interface to the QUADZ receiver and MDS controller software. Test plans and procedures will be written to verify that the messages are correctly received and correlated.

The GPS messages (AMC Only) will be received according to the NIMA 183 - 2.0 standard. An "own-ship-id" will be used to uniquely identify each message for Correlator processing to uniquely identify our "ownship" platform.

4.6.1.2 Initialization

The IOP process will be started by the sys_exec procedure before all the processes except for the IPC process.

4.6.1.3 Inter-process Communications

All QuadZ communications will be sent over the LAN.

4.6.1.4 Error Handling

No error handling processes are required for this functional area.

4.6.2 MSG Functional Area Changes

The following sections describe the items that are changed in order to meet the requirements of this functional area.

4.6.2.1 New Software Libraries

No new software libraries were developed.

4.6.2.2 Modified Software Libraries

No software libraries were modified.

4.6.2.3 New User Interface Windows

No new user interface windows were developed.

4.6.2.4 Modified User Interface Windows

The Comms Config window will be modified to include setting up an Input line as a TCP/IP Client. “Client” was added to the “Input Device” pull-down of the Configuration sub-menu window. Previously all input lines were automatically set as server lines. The Integrated Communications Processing (IOP) however is the server in this architecture, hence the need for the change.

4.7 Integrated Communications Processing Functional Area Design

The Integrated Communications Processing is the first ABI process to start. It creates the shared memory segment, initializes portions of the segment, starts client or server software (described below), and sets up TCP/IP communications ports representing the data streams from the various external data sources (GPS, TRAP, TIBS, TADIL-A). It then waits for either interaction with the ABI control software (connecting to the data sources) or the ABI correlation engine (desiring the data). The IOP program reads data from the various external data sources, does limited processing, and relays the data to the correlation engine. As it reads and writes, the IOP program computes interface health (when last seen, how much) for operator viewing.

4.7.1 Integrated Communications Processing Architecture

The Integrated Communications Processing program consists of a main routine, initialization routines, special processing routines (GPS, TIBS), and access to the standard ABI control routines. Processing consists of initialization calls and a “Do forever” loop that does a sleeping poll for I/O activity. The "sleeping" portion allows the process be awaken every sleep interval (about 100 milliseconds) for timely processing or whenever data is present. On awaking from the sleeping poll, IOP processes any I/O needs, and then does "timely" processing such as interface health or GPS DR. The “Do” loop then continues a sleeping poll again.

4.7.2 IOP Processing Description

The IOP initializes the shared memory segment, loading the IDB and TRAP system definitions
The preference file is accessed and the type of service desired is determined
If a server:
    Clear trigger files and verify that the disk setup for ABI is of the form /abi/disk1-n

```
    Launch whatever programs are needed (data svc) and define access ports for data service ports
    6027-6032.
Do forever:
    Poll the current I/O connection (sockets and ports)
    Returns from poll with either n  I/O events or times out.
    Check for signals (sig term, sig poll) and perform the associated service
        If poll returns no I/O events (time out):
        Check each connection
        Report link time acts for data sources without data for n minutes or inject data to send
        a data sink (a user) else have n I/O events.
For each user:
    If a read I/O event for this fd-
    If a service events - do sout accept and add to user set as a data sink (user)
    Else if GPS  or a data source - read in n bytes - send y data to data sinks associated with serial
    Else a write I/O event - write n bytes
End for each user:
    Periodically check for new/changed I/O connections
    Compute line thru-put
    Check for IDB update and injected data files
```

4.8 Integrated Visual Control (MSTS) Functional Area Design

The visualization process is controlled by a "banner" program that is started when the operator selects ABI visualization. This process reads user customized values and starts up the default visualization programs. If the operator desires to change the layout of the visualization screens, the operator can either attempt to grab the edge of the screen they desire to change, or go through the MSTS banner program which allows the operator to define screen layouts in one of six ways and get a particular layout by pressing one screen button. The MSTS program will be the central point to have other control and configuration programs started for operator interaction. The programs are listed by function under five pull down functional areas (Setup, Display, Threats, Draw, and Utilities). Special buttons will be presented in the MSTS banner for fast, single click operations to get a screen snapshot, define a map centering point, get GPS parameter readout, or to freeze the GPS time and location readout for logging.

4.8.1 Integrated Visual Control (MSTS) Architecture

MSTS consists of a main routine which initializes the user information, builds a banner window of pull down menus and push buttons, starts various other visualization and control programs, adds a one second timer call function to the X window functionality, and then calls the XtMainLoop() function which hands processing over to the X server. As timers decrease to 0 or buttons are pressed, or selections made, various routines within the collection of software making up MSTS (called X callbacks) are called by X to process each event. The routine performs some simple tasks (such as the GPS data pop up), starts a program (radio control, filter control, expert functions), or sends commands to visualization programs to change their format when the operator presses a screen layout button. The timer function updates the time and GPS derived position information displayed. Most functions return to X to allow for other event processing. The System shutdown and Quit functions are two events that do not return to X.

4.8.2 Integrated Visual Control (MSTS) Processing Description

```
Startup X services
If a server:
    Start del-logs to check for files to delete
Check to ensure IOP is running
If a server:
    Delete trigger files
    Launch appropriate radio control program
    Start any local correlators (TIBS, TADIL A, TADIL J)
Else a chart:
    Run rdate to get date and time from server
    Define MSTS window features
    Realize the main widget
    Start the WDBMAP program
    Setup two X timer events
        Call Timer callback each second
        Call Delay callback after 3 seconds
    Continue in the X Mainloop
When Delay callback timer expires:
    Start up appropriate radio control dialogue
When Time callback timer expires:
    Check for new trap filters (drawn)
    Call the appropriate filter program to complete and send to radio
    Check and display any test messages; or program calls from IOP
    Set correct button sensitivity
    Update the random display (GPS)
    Check for dead children
    Check for react of imagery
    Move any dynamic  Bullseyes
    Handle any command files
    Check for Flight Route file changes
    Update (DR) each active flight route
    If moving alert filter defined:
        Set appropriate altitude flag
    Check each active track
```

```
        Set bltnk if necessary
        Compute an alert if too close to a kill ring
```

4.9 Country Outline Visual (WDBMAP) Functional Area Design

When MSTS starts up, it launches the WDBMAP program to visualize where the system is and get a simple situation awareness and visualization layout tool. The WDBMAP program then displays country outlines, rivers, lakes, and internal borders along with user selected AOI lines, route lines, and symbols representing the threats in the shared memory segment database that have been processed by the correlation engine. The WDBMAP program allows the operator to scale all of these features using a variable zoom feature as well as display the extent of the maps, charts, imagery, and DTED over the world outline. The WDBMAP program initially centers the map approximately in the GPS reported position and allows the user to re-center the map based on another defined ground point, a simulated flight, or the map centering flag. The WDBMAP program is also used to display the location and viewing direction of the 3D flying visualization program's point of view as well as satellite ground tracks from orbit visualization.

4.9.1 Country Outline Visual (WDBMAP) Architecture

The WDBMAP program consists of a main routine which initializes X and the data structures needed by WDBMAP and a series of X subroutines called "callbacks". Each callback has an implied X event such as pushing a button or selecting a value. When the X event occurs, the XX mainloop function serializes that event with all other events and calls the appropriate callback routine in the order of the event serialization. Each callback routine is responsible for processing or a passing on its event.

4.9.2 Country Outline Visual (WDBMAP) Processing Description

```
Access shared memory.
Define the window parameters; context, GC, main-widget, default color map
Load the Generic symbol definites.
Load the previous zoom factor configuration
Assume all map types have coverage
Initialize the orbit database
Setup a X  timer to cal Timer callback in 3 seconds
Continue in the X Mainloop

When Timer Callback timer expires:
    Complete if time to re-draw of map
    Compute new re-center location
    Draw moving tracks
    Check Display data (Draws, other area actions)
    If time to re-draw all
        Call Re-draw All
    Else:
        Draw Update
        Poll any Route updates/changes
    Start a new times event to call  Timer Callback in 1 second
```

4.10 ADRG Chart Visual (ADRGDSP) Functional Area Design

This program performs like the WDBMAP program in performing a centered, localized view of the geography of the area while being annotated by AOIs, routes, and threat symbols. The display background for the ADRGDSP program consists of colored navigation charts that have been digitized by DMA (now NIMA) and published in CD-ROM form. The charts come in different families, according to usage. The families used by ABI are:

- Global Navigation Charts (GNC) scaled 1:1,000,000
- Joint Navigation Charts (JNC) scaled 1:500,000
- Operation Navigation Charts (ONC) scaled 1:250,000
- Theater Pilotage Charts (TPC) scaled 1:100,000
- Topographic Line Maps (TLM) scaled 1:50,000

The files on a CD-ROM represent a full scale rendition of the original paper charts as well as a 1/16 scale version for quick overview. The CD-ROM files are copied to the ABI disk as well as a new file generated to show the area at 1/4 scale. These three versions make up the fill scale, mid resolution, and overview charts of an area for a particular chart family.

4.10.1 ADRG Chart Visual (ADRGDSP) Architecture

Like the WDBMAP program, the ADRGDSP program represents a geographic visualization for ABI. It consists of a main routine that performs initialization, database routines that handle the data for the background, a periodic poll timer, a re-draw function, and access to a common drawing package for AOI, route, bullseye, and threat drawing in most visualization environments. As the poll timer event occurs, the ADRGDSP timer routine is called, which determines if the screen is to be redrawn completely (map zoomed, re-centered) or just updated. The appropriate functions are then called to re-draw or update the display, and the ADRGDSP program waits for the next timer event.

4.10.2 ADRG Chart Visual (ADRGDSP) Processing Description

```
Access shared memory
```

```
Define the window parameters:  maid widget, GL, default, color map, resources
Create display window
Load the generic symbols
Set X timer
   X mainloop
When Timer callback time expires:
   Compute if time to redraw
   Compute centering loc.
   Draw map tracks
   Redraw (if time)
Else
   Draw update
   Pole any route update/change
Start a new timer event to call Timer callback
```

4.11 Imagery Visual (EOSATDSP) Functional Area Design

This program performs like the WDBMAP program in performing a centered, localized view of the geography of the area while being annotated by AOIs, routes, and threat symbols. The display background for the EOSATDSP program consists of imagery files that have been geo-registered and processed by other ABI software into tiles and condensed like ADRG charts. The imagery is either black & white imagery from a high resolution satellite (like the French SPOT satellite) or false color multi spectral displays from broad area imagery satellites like EOSAT.

4.11.1 Imagery Visual (EOSATDSP) Architecture

Like the WDBMAP program, the EOSATDSP program represents a geographic visualization for ABI. It consists of a main routine that performs initialization, database routines that handle the data for the background , a periodic poll timer, a re-draw function, and access to a common drawing package for AOI, route, bullseye, and threat drawing in most visualization environments. As the poll timer event occurs, the EOSATDSP timer routine is called, which determines if the screen is to be redrawn completely (map zoomed, re-centered) or just updated. The appropriate functions are then called to re-draw or update the display, and the EOSATDSP program waits for the next timer event.

4.11.2 Imagery Visual (EOSATDSP) Processing Description

```
Access shared memory
   Define window parameters:  main widget, color map, GL, res.
   Create display window
   Load generic symbols
```

```
    Find applicable image
    Setup X mainloop
When Timer expires:
     Compute if time to redraw
     Compute centering loc.
     Draw map tracks
     Redraw (if time)
Else:
     Draw update
     Poll any update changes
Set a new Timer event
```

4.12 3D Flying Visual (FLY) Functional Area Design

The ABI system allows the operator to 'fly' in an area where the ABI system has elevation data called DTED as well as ADRG charts or imagery. With these two data sources (DTED and a 2D geographic representation), a 3D visualization can be computed and viewed. Due to the speed and capabilities of modern graphics systems such as SUN's Creator 3D graphics workstations, these 3D visualizations can be shown in near real time and at speeds comparable to Air Force transport aircraft. This visualization gives the user the sense of looking out of the cockpit window at locations the user has never been to before, with the capability to pan 360 degrees as well as change the viewing altitude and viewing inclination angle.

4.12.1 3D Flying Visual (FLY) Architecture

The FLY program consists of a collection of routines, written in OPENGL that perform functions similar to other 2D visualization programs. There is a main routine which performs initialization and display setup; defines the various event routines to be called in its WIN_INIT () function Call, and the calls an external, windows oriented, mainloop and event processing function. As events occur, the mainloop function calls the various even processing functions defined during initialization to perform keyboard actions, resize actions, mouse actions, button actions, and drawing actions.

4.12.2 3D Flying Visual (FLY) Processing Description

```
Call Win_Configure () to define the window in a motif environment
Open the DTED database
Load all Flyable routes
Load a coverage definition of the ADRG and EOSAT (MST) databases
```

Call WIN_INIT () to define the functions to be called for various window events and define a window visual

Call WIN_GetSize () to acquire the startup window sizes so the appropriate 3D view can be computed

Define the OPENGL perspective, initial aircraft position, threat position and AOI position

Call the windows mainloop function to process all future window events

4.13 Satellite Orbit Visualization (OV) Functional Area Design

The ABI system supports not only visualization of where the operator is geographically but also presents an outer space view of the planet and the estimated location and viewing areas of defined satellites. This situation awareness can then go toward the ability to see communications satellites and have a well populated constellation of GPS satellites for geographic positioning, or the ability of an intelligence satellite to see a desired portion of the earth. The world is then displayed as a rotatable ball with curves representing the path of satellites of interest and line of site cones projected to the earth's surface from the satellite's computed position for limited sight satellites. In addition, ground tracks can be computed and displayed on the WDBMAP visual and timelines can be computed to show when various satellites can see a ground spot.

4.13.1 Satellite Orbit Visualization (OV) Architecture

Like the FLY program, OV presents a 3D view using OPENGL. There is a main routine which performs initialization calls and display setup logic. The window texture map is read and stored. An OPENGL call is made to acquire the active texture map size supported by the host computer. If it is too small (X by X texture map compared to height and width of texture file loaded), the OV program aborts. Otherwise, it calls the mainloop window function (XXapp Mainloop).

4.13.2 Satellite Orbit Visualization (OV) Processing Description

Call WIN- Configure (to define the window in a motif environment
Call WIN-Init () to define the function to call for various window events and to define a window visual
Call WIN-Get Sje () to acquire the startup window sizes so that the appropriate 3D view can be computed.
Define the OPENGL perspective, colors, fonts, and texture map.
Call the window mainloop function to process all future window events.

4.14 Radio Control (ZEBRA) Functional Area Design

With the development of Mnemonics' *ZEBRA* radio, an integrated solution to radio control, cryptographic control, and link processing has been developed for ABI. The radio control function supports simple radio tuning, geographic filter definition and control, and simplified testing. This is one of several radio control methods within ABI. See 4.24 (MATT Radio) for another method.

4.14.1 Radio Control (ZEBRA) Architecture

The ZEBRA process is a partially integrated radio controller for the QUADZEBRA radio. It assumes full and complete initialization and command of the radio, with simple trigger files created to tell external processes of change in the radio configuration or control. All trigger files are created in the /tmp area and denote such events as: "Ready to connect for TRAP", "New TRAP filter(s) Installed", etc.

4.14.2 Radio Control (ZEBRA) Processing Description

```
Initialize the X environment
Initialize (connect to) the radio
Determine which form of functional is to be initially displayed
Access the command pipe for future command
Create the initial window
Create an X timer to evaluate events every second
If should hide:
        Make window invisible
Call XX  Mainloop () to process all future events.
```

4.15 Symbol Control (SYMFILTER) Functional Area Design

Every correlated track defined by the correlation engine has a symbol assigned to it. The symbol graphically identifies the type of radar or object and identifies it (friend, foe, commercial, neutral). These symbol definitions are displayed in each graphical visualization process described and presented by the SYMFILTER program. In the SYMFILTER program, the operator has the opportunity to select those symbols that the operator wants to see. Additional control features include a visualization of the estimated ring of detection ('see me' ring) and/or the lethal capability range of the track ('kill me' ring) is desired. The operator can also request that new tracks of specific types be display using an alert (blink) visual when they are reported out of the correlator. These basic symbol controls allow the operator to determine how and what will on the ABI display. The format of the symbols can also change by using the PREFERENCE program described below.

4.15.1 Symbol Control (SYMFILTER) Architecture

All symbols are initially assumed to be desired when the ABI system starts because we assume that the ABI system will move from aircraft to aircraft and one flight crew to another. This is the **current** filter settings. The SYMFILTER program will read the **recorded** filter settings when first selected (under *Setup->Filters->Symbols*). Operator selections then modify both the **recorded** filter settings and the **current** filter settings.

The process consists of a main routine that builds a symbol button display as well as a speed button lower panel. The program then calls XtMainLoop() to turn over event processing to X. When a button is pressed, the X event processing calls the registered callback function to perform the implied action. Typical actions cause symbols to be selected or deselected. The OK button performs the writing of new **recorded** and **current** filter settings as well as unmanaging the display window (the program is still running). Successive selection of the *Setup->Filters->Symbols* operation will revisualize the window with the settings as the operator left them. The Cancel button action unmanages the window only. The Rings button will present a Threat Rings version of the symbol window with just those symbols identified as "Systems" with their implied detection and threat rings defined in a systems file. Button operations are the same in the Symbol Filter display, except that Cancel and OK cause a return to the Symbol Filter display.

4.15.2 Symbol Control (SYMFILTER) Processing Description

```
Access Shared Memory
Initialize display parameters:  Main-widget, default resources
Check for ring filter, symbol filter, or IDB filter option choice
Get stored values
Build appropriate filter display
XX Mainloop
Callbacks:
        OK- flag symbols for display
            - flag symbols for alerts
            - flag rings for display
            - close window
        Cancel
            - Close window
        Help
            - Displays help window
```

4.16 Area of Interest (AOI) Definition/Control AOI Functional Area Design

There are three basic forms of scalable annotation available in ABI: Areas of Interest (AOIs), Bullseyes, and Routes. AOIs are defined as either circles of a user selected radius or a series of line segments. Both forms require a color selection by the user and do support a simple string annotation additional feature. The AOI program supports the creation, modification, deletion, and visualization control. The definitions and visualization control definitions are kept in files and are maintained from mission to mission.

4.16.1 Area of Interest (AOI) Definition/Control *AOI* Architecture

The AOI program represents a typical data management function within ABI. The program can be called by the MSTS program with specific parameters to complete some action (such as drawing a new AOI) or for general data management. The data management functions include: Edit, Delete, Make Visible, Make Invisible and Create. A standard list of existing AOIs is presented along with buttons representing all of these functions. An AOI file needs to be selected for all but the Create function before a function button is selected. The Create and Edit functions present a special form with fields for the contents of an existing or entry of a new AOI. The operator selects fields, enters in the desired values, and presses the OK button. If all of the values pass validity checks, the AOI is created as a new file and entered into the AOI select list of the original display.

Software organization of this process consists of a main routine that initializes the windowing structure, parses parameters for special initial processing, and then calls XtMainLoop() for Xevent processing. Each button has a callback routine that performs the desired action and then either returns to X(display still active) or closes the window and terminates the program (OK and Cancel). The AOI program is assumed to be used to edit or create a single AOI. Visibility changes can be accomplished on multiple AOIs without the program terminating.

4.16.2 Area of Interest (AOI) Definition/Control AOI Processing Description

```
Access Shared Memory
Initialize display parameters
Check for graphical AOI creation
If not graphical
        present AOI dialog box
Else
        Allow user to draw
        Present AOI dialog box
XT Mainloop
Save Callback checks for valid data, saves
```

```
Display AOI
        symlink selected AOI
        signal AOI update
Remove from Display
        unlink AOI
        signal update
```

4.17 Route Definition/Control (ROUTE) Functional Area Design

Like AOIs, routes are managed by a data management program structured specifically for routes. Like AOI, the ROUTE program has two faces, the file management and visibility control section and the Create/Edit section where specific parameters making up a route can be entered. Additionally, routes are **ASSUMED** to be flown with a takeoff time and estimated speeds and altitudes to fly the route by. When a route file is first seen by the visualization software, it will be flown if the start time is in the past.

4.17.1 Route Definition/Control (ROUTE) Architecture

Software organization of this process consists of a main routine that initializes the windowing structure, parses parameters for special initial processing, and then calls XtMainLoop() for X event processing. Each button has a callback routine that performs the desired action and then either returns to X (display still active) or closes the window and terminates the program (OK and Cancel). The Route program is assumed to be used to edit or create a single Route. Visibility changes can be accomplished on multiple Routes without the program terminating.

4.17.2 Route Definition/Control (ROUTE) Processing Description

```
Access Shared Memory
Initialize window parameters, main-widget, resources
Initialize default directory
Check for graphical Route creation
If not graphical:
        present Route list
If graphical:
        allow user to draw
        then present Edit window
XX mainloop
```

4.18 Preference Selection (PREFERENCE) Functional Area Design

The user preference selections are controlled by two major programs: Pref and wx-pref. The PREF program handles the drawing control and special events options controllable by the operator. Such things as live widths, symbol set selection and size, and history depth are set by the PREF program. The Wx-PREF program controls the special preference valves associated with viewing denied area weather tracks that are received over TRAP: Such things as which weather values are critical to the aircraft (temp, wind speed, etc.) are selected by the user for various colorized visualization.

There are two other programs that modify user preferences and are discussed elsewhere. Purge to set track and contact purge times and symbol filters to set attitude and alert times. There are also many preferences which do not have MMI support and are used to configure the software. These preferences are set using a UNI text editor and a knowledge of what each preference controls.

4.18.1 Preference Selection (PREFERENCE) Architecture

Each of these programs consists of a main routine that initializes the X environment, creates graphical widgets for each preference value to control, populates these widgets with the current preferences value, and calls XT Mainloop.

If the operator changes a widget value, the widget display is updated but not the corresponding preference value or file. Only on the operator's selection of "OK" are the widgets read, matched to their preference value and stored both in memory and to disk for the next session.

4.18.2 Preference Selection (PREFERENCE) Processing Description

```
Access shared memory
   Initialize display parameters
   Get stored preferences
   Build display
   XX mainloop
Callbacks
   Set the preference value
   Save values to preference file
```

4.19 Satellite Selection (SATSELECT) Functional Area Design

The SATSELECT program works with orbit visualization to allow the user to identify satellites of interest, associate them into groups, and select individual satellites and/or groups for orbit visualization. SATSELECT also allows the user to select the colors used to display the selected satellites and their sensors.

4.19.1 Satellite Selection (SATSELECT) Architecture

The SATSELECT program consists of a main routine which initializes and defines the main control window. The control window displays all the satellites, all the defined groups, the selected satellites, and buttons to change the selections, group definitions, and OV visualization properties/colors. The main routine builds the window, loads the existing satellites and group definitions and then calls XX mainloop to await window events.

4.19.2 Satellite Selection (SATSELECT) Processing Description

```
Initialize the X interface
Define the main-widget
Load application resources
Ensure we are the only SATSELECT running
If not:
    Tell older program to “raise” self
    Exit after telling older program
Build the main control window
Load all defined satellites
Load all defined groups
Load the visualization defaults
Load the definition of which satellites are active
Set the control button sensitivity based on what has been loaded
Setup an X timer to check for “raise” events
Call XX Mainloop () to manage the window and process all future X events
```

4.20 Track and Communications Status (STATUS) Functional Area Design

The STATUS program periodically polls the counts of tracks, contacts and I/O status and displays those values in three ways - from very simple to very complex. The user is able to select how detailed the data is presented by selecting “MORE” for more detailed or “LESS” for less detailed.

4.20.1 Track and Communications Status (STATUS) Architecture

Based on the configurations defined in preferences, the STATUS display looks at selected I/O ports and tables to define what is in the system These simple counts are displayed in a tabular fashion for periodic or instant viewing of the internal activities within the system. The most advanced tabulation includes the sizes and current usage of various internal tables. On a periodic basis (currently hard coded at 5 seconds), the counts of the viewed objects are recomputed and displayed.

4.20.2 Track and Communications Status (STATUS) Processing Description

```
Access Shared Memory
Determine track source
Initialize Data
Initialize Display Parameters
Build display
Set X Timer
XX Mainloop
When Timer Expires:
        Update status displays
        Set new timer event
```

4.22 Archive Management (ARCHIVE) Functional Area Design

The ARCHIVE program gives the user tape archive capabilities to record log files and screen snap shots.

4.22.1 Archive Management (ARCHIVE) Architecture

The program consists of a main function which defines the file list along with buttons to delete or archive the selected files. User selection and choice of button determines what will occur. The operator can delete or archive. When either is selected, the process is performed and the program terminates.

4.22.2 Archive Management (ARCHIVE) Processing Description

```
Access shared memory
Initialize display parameters
Create window
XX mainloop
Callbacks
        Tar archive files
        Delete archive files
```

4.23 Exercise Input (EXERCISE) Functional Area Design

The EXERCISE program works in concert with the MESSAGE program to allow for scripted injection of contact reports of interest.

4.23.1 Exercise Input (EXERCISE) Architecture

The EXERCISE program is the script input point for the user, where selected contact report of predefined locations can be injected into the system. The EXERCISE Program defines the script, and the MESSAGE program reads the script and injects contacts at the appropriate time.

4.23.2 Exercise Input (EXERCISE) Processing Description

```
Access Shared Memory
Initialize Display parameters
Create Display of exercise list
XX Mainloop
Callbacks
   Delete
      Delete exercise file
Activate
   Unlink old exercise file
   Link new exercise file
Quit
   Exit window
Help
   Create Help window
Edit Create
    Create display
    If Edit load points
```

4.24 MATT Radio Functional Area Design

With the introduction of the MATT radio in addition to the QUAD NET and QUAD ZEBRA radios, a new radio program design was developed. In processing the MATT, there are two programs responsible for control and tuning: MATT_CTRL and MATT_RADIO. They operate as a foreground and background pair, sharing data through a separate shared memory segment. The MATT_CTRL program is the operator interface to the capabilities of the MATT and where specific functional service is selected. The MATT_RADIO program decodes the functional service request into one or more MATT commands and records the entire communication interaction with the MATT radio in a portion of the shared memory segment mentioned above. This allows the operator and MATT_CTRL to see how the MATT command is working as well as any report output by the MATT radio. The processing is written in an asynchronous fashion so that either ABI or the MATT can be powered up first.

4.24.1 MATT Radio Architecture

Both programs are started by MSTS when the Start_ABI button is pressed. MATT_RADIO attempts to connect to the control (Maintenance) port of the MATT radio and listen to what the MATT has to say. Each phrase (line) sent by the MATT radio out the maintenance port implies what is happening or has happened to the radio. The MATT_RADIO program attempts to classify each phrase and feedback to the user and MATT_CTRL what stage of usability the MATT radio is currently in. As it takes over six minutes to boot up the MATT radio and over twenty minutes to re-load the FLASH EPROM software and table definitions, the MATT_RADIO program has lots of time to wait and analyze. It also can tell the MATT_CTRL program "Not Yet" in terms of attempts to command/control the radio while the bootup or software load process is being performed.

The include file matt_gbl.h defines fifteen different MATT_STAT values for the various phrases and conditions that the MATT_RADIO program can detect from the MATT radio chatter. It also defines the eleven commands (MATT_HL_CMD) that the MATT_CTRL program can request the MATT_RADIO program to perform.

This status and control information is periodically reviewed by the MATT_CTRL program to enable or disable the action buttons present on the MATT_CTRL program's various windows. Long-winded commands (i.e., "Activate the radio") are broken into pieces and each piece is sent. The command results are watched until the MATT_RADIO program says that the command is done, and then another request is then presented to the MATT_RADIO program.

Of special architecture note are the files and directories in /abi/share/matt:

The file named "services" has a list of the supported link services. It is assumed that there are sub-directories of the directory "satellites" that mention their services by name and define the selectable frequency settings in a file called "freq4svc_sat" for each service. This file defines both frequencies and names of each link to use to find the other tuning parameters for that specific link. As an example, assume the "services" had an entry for "TRAP". Inside the satellites directory would be a sub-directory called “TRAP”. Inside the “TRAP” sub-directory would be a file “freq4svc_sat” with entries such as:

```
ATLANTIC=345.678

PACIFIC=452.195
```

“ATLANTIC” and “PACIFIC” now name specific TRAP links whose complete link parameters should be found in a file named “.default_ATLANTIC” and “.default_PACIFIC”. This series of files and directories allows for easy configuration and growth of link and parametric definition.

In addition, the /abi/share/matt directory contains directories for the definition of CI filters (inside ci-filters), unknown signals filter (unk-sig-filters), TIBS filters (sids-filters), categories of interest filters (coi-filters) and the control of unknown and category of interest filters on a geo-filter basis (inside geo-filters.)

4.24.2 MATT Radio Control Processing Description

```
MATT Radio:
      Access MATT Shared Memory, ABI Shared Memory
      Get I/O definition of link to use to talk to MATT
      Open MATT serial interface with XON/XOFF  flow control
      Install the MATT Control Connection in the IOP status list
      current_action=idle
Do forever
      If current_action not previous action:
        Tell user of new MATT status
      Poll all connected fd’s
      If received, terminate signal:
        Close everything and exit
      If a poll timeout no actually for 1 second:
        Compute how late MATT may be in completing a command

        If request to load MATT SW:
          Call Look4_work to send first command
        Analyze MATT status based on current MATT status:
          If idle:
          Look for new work
        Else a poll with some I/O event - read data from MATT or write for each Fd in the poll list
          If a read or connect event:
            Service the read or connect
          If a read of data from the MATT:
            Copy and analyze all data received so far
            All complete lines go to a status area
             Incomplete lines are analyzed for possible request for a value.  (MATT
            command responses are of the form variable:  where user is supposed to react
            to the value and meaning of the variable ad later in a response)
        If a query value:
            If a CFL command and response to tabulate all values:
               return CR or $ depending on if a
                    CFL variable  present
                    Else if current command has options:
                            Search for operand match
                    If found - send response:
                    mark this opened motel as complete and mark this phrase as reported to.
                    If not responded to:
                           Check special response list
                    If a special response:
                            Send appropriate character

           Else - Send a CR - Generic response:
               If a read or connect event:
               End
               If a write event:
                    Write out as many bytes as allowed or as many bytes as are left in
                            external message
                    If all bytes sent:
                            Clear write poll flag
                    Else: Set write poll flag-will test for ability to write

MATT_CTRL:
        Access MATT private and ABI shared memory
        Initialized X
        Create the Initial Control window
        Load all data service types
        Load each service's satellite definitions
        Load the list of defined Geo filters
        Load the list of CI filters
        Load the list of TIBS filters (SIDS)
        Realize the main widget
        Start an X timer to call Timer Callback
        Call XX mainloop to service future X events

When the X timer expires:
        Set button sensitivity depending on which window is active
        If in process of activating the MATT:
                Do next command until "did_real_work"
                   Switch on the next command
                   If values to command MATT:
                        Decode into a MATT command
                    Send to MATT
                    Set flag_did_real_work
                Define the next command to switch on - end of switch
                If did real work:
                    Describe actions to user in a status message -
                    End do:
If all work is done:
    Write "activation complete" to user
    Wait 5 seconds, then terminate
```

4.25 Angle to Satellite Computation (FINDGEOSAT) Functional Area Design

The FINDGEOSAT program computes the azimuth and elevation a ground point to a geo-synchronous satellite .

4.25.1 Angle to Satellite Computation (FINDGEOSAT) Architecture

The program has four assumed satellite locators (West 23, West 100, East 72 and East 172) as well as a user selectable input option. The user defines where he is (via GetPoint), which satellite he is attempting to view, then presses "Compute" for an angular computation of line of sight. Negative or near zero angles are usually a sign that the satellite cannot be seen from that point on the world.

4.25.2 Angle to Satellite Computation (yyy) Processing Description

```
Access Shared Memory
Initialize X
Get X resource values
Define the operator selectable widgets and the window
Realize the window
Call XX Mainloop to process all future X events
```

4.26 Expert Functions Functional Area Design

Those functions considered to be outside of normal operations are lumped under the "Expert Functions" banner. This collection of programs has more human interaction then does the rest of ABI and assumes that the user possesses the required level of expertise.

4.26.1 Expert Functions Architecture

The expert functions supported are tabulated in a program table inside of MSTS.C (called children). Those programs tabulated after the adrgimp program are considered to be expert functions and are presented in the expert list of the callbacks.c in MSTS. Chart imports and deletes are specified here as are expert radio control operators (MATT filters, Quad Zebra expert function access).

Communications parameters can be checked and modified here and expert functions can be protected by a password. Finally, a detailed, internal look at what the system contains can be seen by running the expert functions Communications status option. The sizes and populations of numerous tables can be viewed as well as detailed I/O throughput information.

4.26.2 Expert Functions Processing Description

The selected program initiates itself and presents the user with a basic MMII

The user makes choices and the MMI performs the action without destructive action safety: i.e., when the operator selects DTED DELETE, specifies a range of Latitudes and Longitudes to delete, and then presses OK, no other requests are made to the user to be sure that the deletes are desire.

4.27 Client Workstation Support (DATASVC, CLIENTSVC) Functional Area Design

Where more than one ABI system is desired the workstations can be arranged with one workstation as the server and the remaining workstation as clients.

4.27.1 Client Workstation Support (DATASVC, CLIENTSVC) Architecture

The DATASVE program operates as a TCP/IP data pusher to any CLIENTSVC programs that connect. As changes are made in the server, copies of the data are sent to all clients. This service is only for the data tabulated in GLOBAL memory tables and does not include user preferences or symbol filters.

4.27.2 Client Workstation Support Processing Description

```
Access Shared Memory
Define a list of current readers, current SVC ports and work Queues.
Attempt to allocate the TCP/IP ports associated with various services -
Do forever:
        Poll the service ports and current user connections
        If received SIGTERM signal:
                Disconnect from everyone and Quit program.
        If any single point database has been updated:
                Add to list of tables to send to all client SVC programs.
        If updates in multi-row tables:
                Add  as n ran updates to list of table to send all client SVC programs.
        If poll data from any connects:
                Perform read or write as inferred in poll results
End do forever:
In the CLIENTSVE program:
Access Shared Memory
Determine which machine is the server

Do forever:
        If terminate signal:
                Close connections and exit
        If not yet connected to server:
                Attempt connection to server
                If connection, get ready to read all table updates
                Else (connected to data server)
                Read a segment of a data command -
                either the common header (CLNT_SRVR_HDR) or the data that follows a
                particular message
                When completely, read header and data
                        Switch on message ID in header and perform selected table update, delete, or
                control operation
        Reset for next CNTL_SRVR_HDR read
```

4.28 System Shutdown Functional Area Design

When the user is finished with ABI, a complete system shutdown is performed. The function/script in /abi/bin/msts_shutdown is performed so that power can be turned off without corrupting data or disk.

4.28.1 System Shutdown Architecture

Currently, the MSTS-shutdown script runs SYNC followed by the system command HALT, which has been coped and renamed /abi/bin/msts_halt. Other forms of shutting down the system can be performed by changing either MSTS-shutdown or MSTS-halt.

5.0 Acronyms

2D	Two Dimensional
3D	Three Dimensional
AA	Automatic Associator
ABI	Airborne Broadcast Intelligence
AOI	Area of Interest
ADRG	ARC Digitized Raster Graphics
AMC	Air Mobility Command
CCS	Combat Intelligence System (CIS) Core Software
CD	Compact Disk
CDE	Common Desktop Environment
CD-ROM	Compact Disk Read Only Memory
CIS	Combat Intelligence System
CPU	Computer Processing Unit
CSCI	Computer Software Configuration Item
DMA	Defense Mapping Agency
DOD-STD	Department of Defense Standard
ELINT	Electronic Intelligence
GB	Gigabyte
GNC	Global Navigation Charts
GPS	Global Positioning System
GUPS	Global Uninterruptable Power Supply
IDB	Integrated Data Base
I/O	Input/Output
IOP	Integrated Communications Processing
IPC	Inter-Process Control
JCS	Joint Chiefs of Staff
JNC	Joint Navigation Charts
LAN	Local Area Network
MHz	MegaHertz
MMI	Man Machine Interface

MSI	Multi-Spectral Imagery
MSTS	Multi Source Tactical System
NRT	Near Real Time
OM	Object Manager
ONC	Operation Navigation Charts
PPU	Protocol Processing Unit
RADINT	Radio Detection and Ranging Intelligence
RAM	Read Access Memory
RF	Radio Frequency
SIDS	Secondary Imagery Dissemination System
SIGINT	Signal Intelligence
SDD	Software Design Document
TIBS	Tactical Information Broadcast System
TBMCS	Theater Battle Management Core Systems
TOR	Time of Receipt
TLM	Topographic Line Maps
TPC	Theater Pilotage Charts
TTA	Tactical Information Broadcast (TIBS) TADIL Associator
U	Unclassified
UIF	User Interface
UHF	Ultra-High Frequency
USMTF	U.S. Message Text Formatting

Appendix A Requirements Traceability

ABI REQUIREMENTS

User inputs from the 24 November 1997 ABI Technical Interface Meetings (TIM) in Fairfax, VA are included in the requirements matrix and test cases. Inputs for software improvements are organized into the following topic areas and include a list of requirements by User ID Tracking Numbers.

This Appendix lists the requirements and test cases applicable to the ABI system. Functional areas have been defined for ABI software development and are listed below. Test cases identify the functional area, test case number, test case name, and test objective for each test. Specific test procedures are identified in the ABI Software Test Document (ABI-AWACS-U-NA-N-001), 19 January 1998 (Revision 4).

PRF - Performance

ENV - Environment

DTB - Database

UIF - User Interface

MSG - Messaging

DSP - Displays

User inputs from the 24 November 1997 TIM are included in the requirements matrix and test cases. These inputs for Software Improvements are organized into the following topic areas and include a list of requirements by User ID Tracking Numbers.

1) More informative displays	C15, C51
2) Easier Capabilities	B04, C17, C50, C36, S05, S11, S13, S14
3) Relating threats to current position	All Deleted at 11/24/98 TIM.
4) Better feedback to user	C03, S10
5) Basic capabilities needing improvements.	C04, C47, D22, D24, S12

The prefix code defining the originator of the user requirements is listed below.

Axx	AWACS suggestions
Bxx	AWACS suggestions not usable by AWACS but useful functions
Cxx	AMC suggestions
Dxx	Discrepancies from AWACS and AMC demos

Sxx Contractor suggestions

Table 1 identifies the ABI requirements and Table 2 provides a cross-reference of ABI requirement to the test cases which satisfy them.

Table 1 Requirements

Requirement Number	Requirement	Test Method	Test Case
PRF-001	ABI shall receive an input of 30 contacts per second and no more than 5 seconds shall elapse from contact receipt to map display.	Test	1.1
PRF-002	ABI shall maintain 24K tracks at any one time and 50K contacts in 24 hours.	Test	1.2
PRF-003	ABI shall refresh the map at the rate of 5 maps per second.	Test	1.1
PRF-004	ABI shall distribute displays and processing over multiple workstations and multiple processors on the same workstation.	Defer	1.3
ENV-001	ABI shall be the single point of entry for Quad Zebra receiver control output.	Defer	5.2-5.5
ENV-001 A	ABI shall be the single point of entry for MATT receiver control output.	Test	6.3
ENV-002	Solaris 2.5 shall be the operating system used for the ABI system.	Observe	2.2
ENV-003	ABI shall use the CDE/Triteal desktop for its user workstation interface.	Observe	2.3
ENV-004	ABI shall address the following issues: startup/initialization auto-upload of an NRT database for air-borne system GPS system time synch configurations issues, error logging, etc.	See Below	N/A
DENV-004-01	ABI background processes shall be started upon user login.	Test	7.0
DENV-004-02	ABI shall not require the user to logon when starting ABI.	Test	7.0
DENV-004-04	ABI shall allow the user to set the system time.	Defer	2.7
DENV-004-05	ABI disk space shall be properly partitioned.	Observe	2.8
ENV-005	ABI shall run on a SUN Sparc Ultra2/200 MHz processor.	Defer	2.9
ENV-005 A	ABI shall run on a SUN Sparc Ultra2/300 MHz processor.	Defer	2.9
DTB-001	ABI Displays shall interface with the ABI database to update the track picture.	See Below	N/A

Requirement Number	Requirement	Test Method	Test Case
DDTB-001-01	New contacts, groups and tracks shall be sent to the ABI Displays.	Test	3.1
DDTB-001-02	Groups and tracks shall be sent to the ABI Displays.	Test	3.2
DDTB-001-03	Delete notifications shall be sent to the ABI Displays.	Test	3.3
DDTB-001-04	Delete notifications from ABI Displays shall be processed by the ABI Data Fusion Engine.	Test	3.4
DTB-002	CIS shall provide the capability to take a save NRT on the ground for transfer to ABI onboard.	Test	7.4
DDTB-002-02	CIS shall provide the capability to write an NRT snapshot to tape.	Test	7.5
DTB-003	ABI shall allow filtering of data going to ABI Displays.	Test	6.4
UIF-003	ABI shall integrate 3D imagery into ABI displays/maps.	Defer	4.3
UIF-004	ABI shall provide the capability to load an IDB and NRT extract.	Test	7.1-7.4
MSG-001	ABI shall receive QuadZ output - TDDS, TOPS, TIBS, TADIL-A.	Defer	5.2-5.5
MSG-001 A	ABI shall receive MATT output - TDDS, TOPS, TIBS, ODP.	Test	6.3
DMSG-001-01	ABI shall receive data from a LAN connection.	Defer	5.2-5.5
DMSG-001-02	ABI shall receive and process TDDS data from a LAN connection.	Defer	5.2
DMSG-001-03	ABI shall receive and process TIBS data from a LAN connection.	Defer	5.3
DMSG-001-04	ABI shall receive and process TADIL-A data from a LAN connection.	Defer	5.4
DMSG-001-05	ABI shall receive and process TOPS data from a LAN connection.	Defer	5.5
MSG-002	ABI shall parse GPS messages.	Defer	5.6
DSP-001	ABI shall provide a smooth scrolling digital map centered on ownship location as represented by GPS input.	Defer	6.5-6.47
DSP-002	ABI shall provide moving alert filters based on ABI aircraft location.	Defer	6.4-6.35
DSP-003	The ABI Displays function shall provide an interface for the user to perform delete functions.	Test	6.14-6.16
TBMCS SP0 1.1	Demonstrate upload capability of data from TBMCS using CIS/AA to the ABI System. (Same requirement as DTB-002, DTB-002-02, and UIF-004).	Test	7.1-7-4
TBMCS SP0 1.2	Demonstrate correlation capability of ABI	Test	AA-04
AWACS SPO 2.1a	All testing will be performed on AWACS Standalone hardware	Observe	N/A

Requirement Number	Requirement	Test Method	Test Case
AWACS SPO 2.1b	A MATT with external antenna will be connected to the AWACS BI processor for tests.	Observe	N/A
AWACS SPO 2.2	All testing will be performed using an agreed to test plan and procedures. All testing for buy off will be performed using a structured, step-by-step process in the presence of Government witnesses recording observations.	Observe	STD
AWACS SPO 2.3	Full start-up and initialization process beginning with power-on (radio, processor, and crypto) and loading of hard drives and other media.	Observe	N/A
AWACS SPO 2.4a	MATT control from the BI processor in all modes to include radio initialization and filter settings for TDDS, TIBS, and TADIXS-B (OPB-TOPS)	Test	6.3
AWACS SPO 2.4b	The ability to download MATT software from the removable hard drive through the processor.	Test	6.3
AWACS SPO 2.4c	The ability to download files from the removable hard drive that define MATT filter settings.	Test	6.1
AWACS SPO 2.5	Simultaneous operation in TDDS, TIBS and TADIXS-B (OBP-TOPS).	Test	6.3
AWACS SPO 2.6	MMI changes requested by the AWACS SPO.	See Below	N/A
AWACS SPO 2.6a	(A02) - Display parameters in Freq, PRI, PW order with ELNOT at end of line or on next line.	Test	6.15
AWACS SPO 2.6b	(A09) - Selectable (smaller) icons.	Test	6.7
AWACS SPO 2.6c	(B04) - User selectable object(s) for Bullseye or compass rose centering, Threat Alert, track recording, and GPS like position readout, map centering, and 3D flying.	Test	6.18
AWACS SPO 2.7	Display and discrimination of both real and exercise tracks.	Test	6.14
AWACS SPO 2.8	User selectable display of entities in the IDB.	Test	AA-
AWACS SPO 2.9a	One Day Orientation Course.	Observe	N/A
AWACS SPO 2.9b	Software User’s Manual (SUM).	Observe	N/A
AWACS SPO 2.9c	Software Test Description (STD).	Observe	N/A
AWACS SPO 2.9d	Software Design Document (SDD).	Observe	N/A

Requirement Number	Requirement	Test Method	Test Case
AWACS SPO 2.9e	Year 2000 Assessment.	Observe	N/A
AWACS SPO 2.9f	ABI_AA ICD change documentation.	Observe	N/A
AWACS SPO 2.10	Test Report. The marked up STD will suffice as long as it documents the test results and performance baseline.	Observe	N/A
AMC 3.1	Demonstration Plan. See AWACS SPO 2.2.	Observe	STD
AMC 3.2	Threshold Requirements.	Test	STD
AMC 3.3	Verification of Test Functionality.	Test	STD
AMC 3.4	MMI changes requested by the AMC.	See Below	N/A
AMC 3.4a	(C03) - Reporting on radio channel basis of possible trouble with radio.	Test	6.32
AMC 3.4b	(C04) - Draw Kill & See Me rings on map when track center off map.	Test	6.39
AMC 3.4c	(C15) - Add a definition pop-up in the TDDS symbol filter display to describe each symbol/system.	Test	6.4
AMC 3.4d	(C17) - Corridor drawing made easy; add possible highest point (s) within identified.	Test	6.20
AMC 3.4e	(C36) - Manual entry of threats using system names.	Test	6.25
AMC 3.4f	(C47) - Add automatic robustness to system initialization for file system integrity checks.	N/A	
AMC 3.4g	(C50) - Rectangular AOI draws made easy.	Test	6.19
AMC 3.4h	(C51) - User selectable velocity vector presented on GPS symbol to show heading.	Defer	6.40
Demo Discrep 4.1	(D22) - Fix line draws which cross map.	Test	6.39
Demo Discrep 4.2	(D24) - Fix TIBS age computation.	Defer	
Contractor 5.1	(S05) - New maps centered using last centering operation.	Test	6.39
Contractor 5.2	(S10) - Display :00 instead of blanks in status for 0 time.	Test	6.11
Contractor 5.3	(S11) - Augment WDBMAP functionality during coverage pulldown.	Test	6.34
Contractor 5.4	(S12) - Remove TIBS query command (net chatter).	Observe	N/A
Contractor 5.5	(S13) - Track select highlight changing as step through tracks.	Test	6.15
Contractor 5.6	(S14) - Augment WDBMAP functionality during coverage pulldown.	Test	6.34

Table 2 maps the requirements to the test cases which satisfy them.

Table 2 Requirements - Test Case Cross Reference

Requirement Number	Test Case Number
PRF-001	1.1
PRF-002	1.2
PRF-003	1.1
PRF-004	1.3
ENV-001	5.2- 5.5
ENV-001 A	6.3
ENV-002	2.2
ENV-003	2.3
ENV-004	See Below
DENV-004-01	7.0
DENV-004-02	7.0
DENV-004-04	2.7
DENV-004-05	2.8
ENV-005	2.9
ENV-005 A	2.9
DTB-001	3.1- 3.4
DDTB-001-01	3.1
DDTB-001-02	3.2
DDTB-001-03	3.3
DDTB-001-04	3.4
DTB-002	7.4
DDTB-002-02	7.5
DTB-003	6.4
UIF-003	4.3
UIF-004	7.1-7.2
MSG-001	5.2- 5.5
MSG-001 A	6.3
DMSG-001-01	5.2- 5.5
DMSG-001-02	5.2
DMSG-001-03	5.3
DMSG-001-04	5.4
DMSG-001-05	5.5

Requirement Number	**Test Case Number**
MSG-002	5.6
DSP-001	6.5-6.47
DSP-002	6.4-6.35
DSP-003	6.14-6.16
TBMCS SPO 1.1	7.1-7.4
TBMCS SPO 1.2	AA-04
AWACS SPO 2.1a	Observe
AWACS SPO 2.1b	Observe
AWACS SPO 2.2c	Observe
AWACS SPO 2.3	Observe
AWACS SPO 2.4a	6.3
AWACS SPO 2.4b	6.3
AWACS SPO 2.4c	6.1
AWACS SPO 2.5	6.3
AWACS SPO 2.6a (A02)	6.15
AWACS SPO 2.6b (A09)	6.7
AWACS SPO 2.6c (B04)	6.18
AWACS SPO 2.7	6.14
AWACS SPO 2.8	AA-03
AWACS SPO 2.9a	Observe
AWACS SPO 2.9b	Observe
AWACS SPO 2.9c	Observe
AWACS SPO 2.9d	Observe
AWACS SPO 2.9e	Observe
AWACS SPO 2.9f	Observe
AWACS SPO 2.10	Observe
AMC 3.1	Observe
AMC 3.2	STD
AMC 3.3	STD
AMC 3.4a(C03)	6.32
AMC 3.4b (C04)	6.39
AMC 3.4c (C15)	6.4
AMC 3.4d (C17)	6.20
AMC 3.4e (C36)	6.25
AMC 3.4f (C47)	N/A
AMC 3.4g (C50)	6.19
AMC 3.4h (C51)	6.40
Demo Discrep 4.1 (D22)	6.39
Requirement Number	**Test Case**

	Number
Demo Discrep 4.2 (D24)	Defer
Contractor 5.1 (S05)	6.39
Contractor 5.2 (S10)	6.11
Contractor 5.3 (S11)	6.34
Contractor 5.4 (S12)	Observe
Contractor 5.5 (S13)	6.15
Contractor 5.6 (S14)	6.34